NYU

Lecture 10:
Transformer & LLM Accelerators

Notes

- Midterm grade will post tonight
- Meeting with the project teams next week

Recap

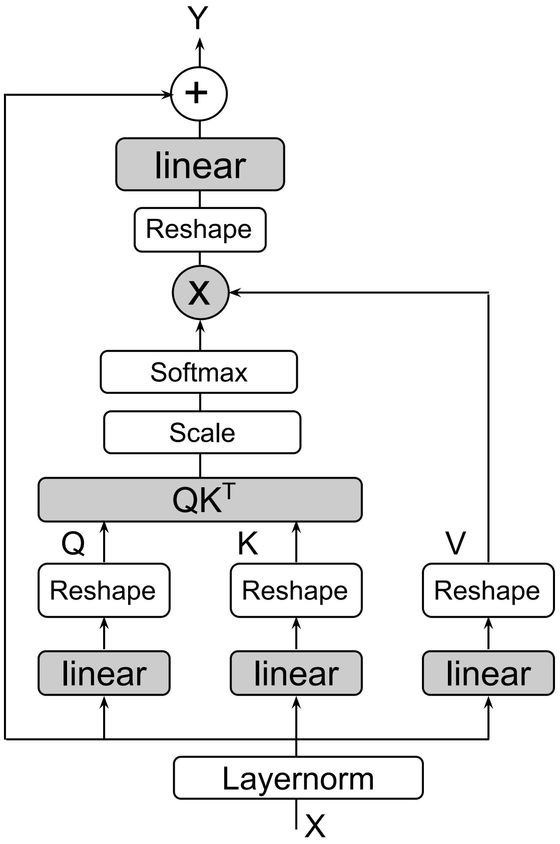
- Convolutional operation conversion
- Hardware architecture of CNN accelerator
- Systolic array
- Popular CNN accelerator design
    - SpAtten
    - EdgeBert
    - Olive

Topics

- Matrix Multiplication with Transposition
- Hardware design for Nonlinear Blocks
- System optimization of LLMs
- Popular transformer accelerator design
  - SpAtten
  - EdgeBert
  - Olive

Self-Attention Block

- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x’, denoted as: Query (Q), Key (K), Value (V).
    - $(B,L,E) \times (E \times E) \rightarrow (B \times L \times E)$
- The second step in calculating self-attention. This will compute the attention score between each pair of input tokens.
    - $QK^T \rightarrow (B, L \times E) \times (B,E \times L) \rightarrow (B, L \times L)$
- Scale and normalize the score using softmax.
    - $\mathrm{Softmax}(QK^T) \rightarrow (B, L \times L)$
- Multiply each value vector by the softmax score.
    - $\mathrm{Softmax}(QK^T) \times V$
    - $(B, L \times L) \times (B, L \times E) \rightarrow (B, L \times E)$
- Pass the result to the linear layer, sum with the input.

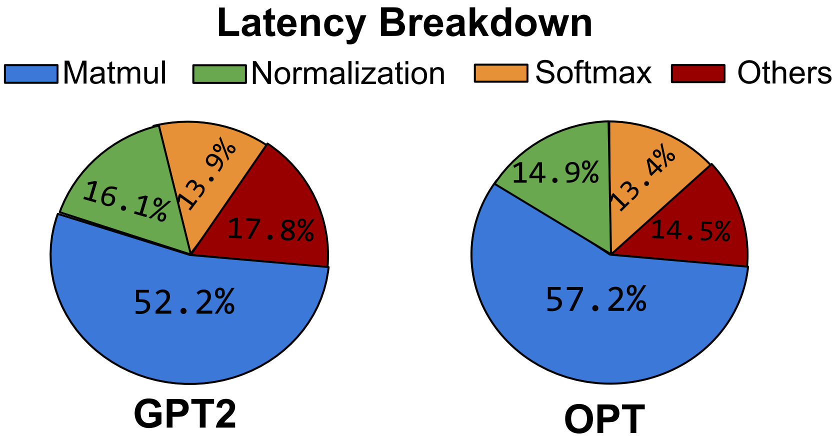


Operations Other than Multiplications

- Transposition
- Nonlinear operations
  - Softmax
  - LayerNorm
  - GeLU

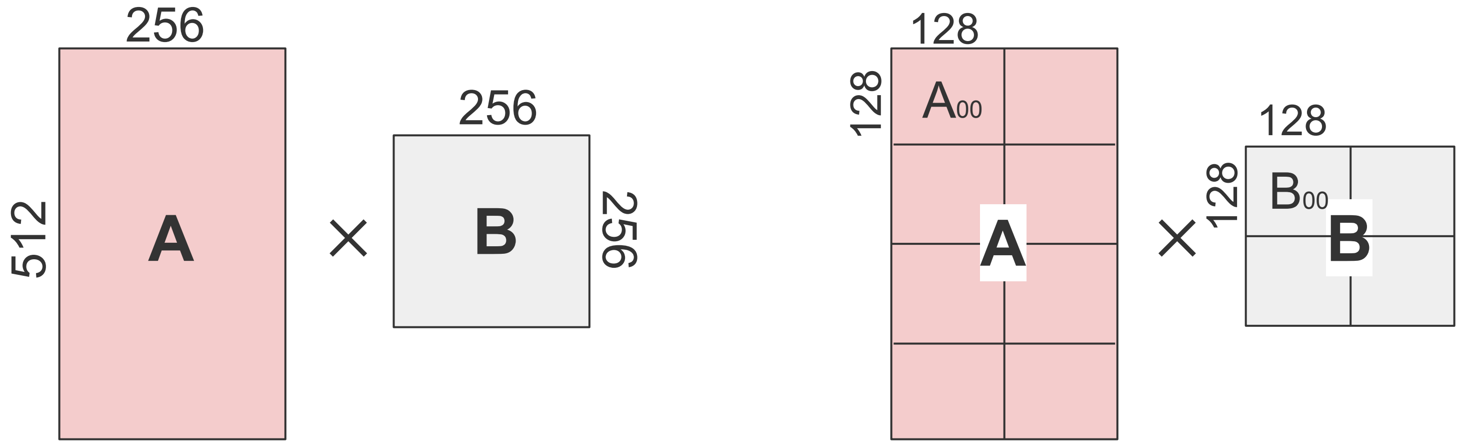
NYU SAI LAB

Breakdown on Computational Cost



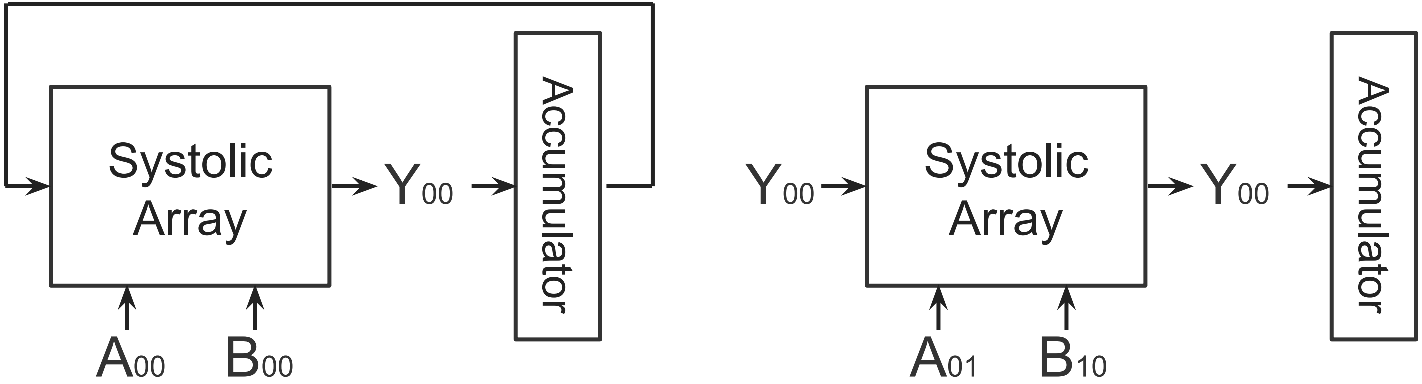
- Matmul still contributes to majority of the overall latency.
- Nonlinear operations are not negligible.
- Also other operations (e.g., transposition) also contributes to a great portion of the overall latency.

Matrix Multiplication



- The large matrix operands are first partitioned into tiles that can fit the size of the compute core.

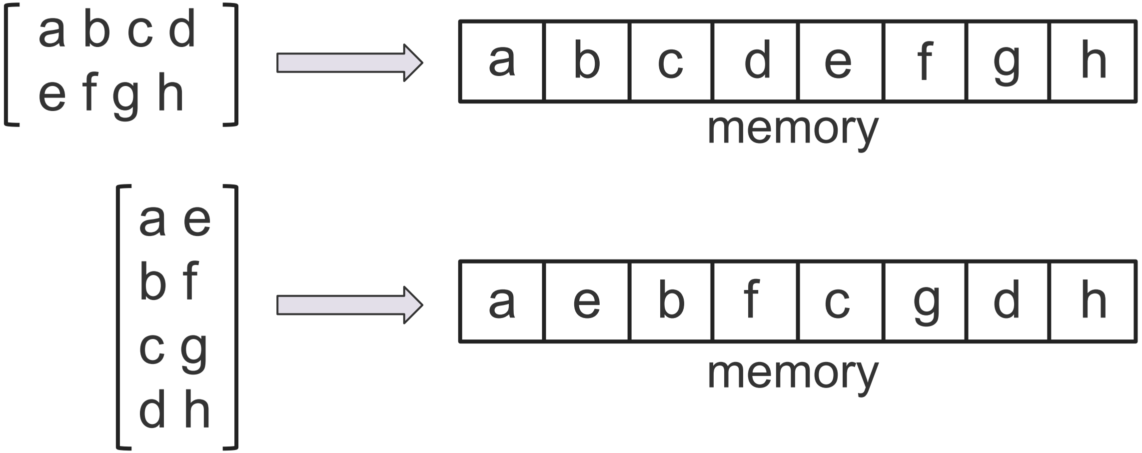
Matrix Multiplication



- The large matrix operands are partitioned into tiles that match the compute core's capacity, after which multiplication and accumulation are executed on a per-tile basis.
- However, sometimes the transposition operations are also required → QK^T

In-Place Matrix Transposition

- In-place matrix transposition refers to the process of transposing a matrix directly within its existing memory space, requiring only a minimal amount of extra storage.



$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix}$$

a	b	c	d	e	f	g	h

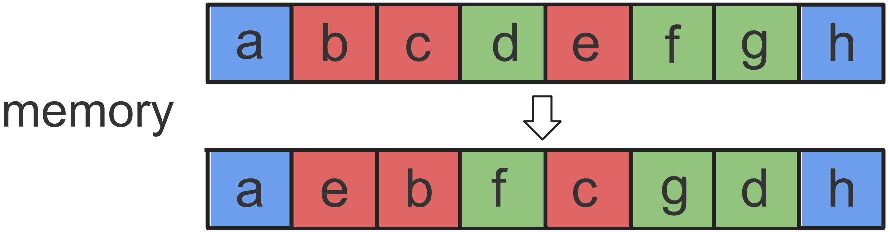
memory

$$\begin{bmatrix} a & e \\ b & f \\ c & g \\ d & h \end{bmatrix}$$

a	e	b	f	c	g	d	h

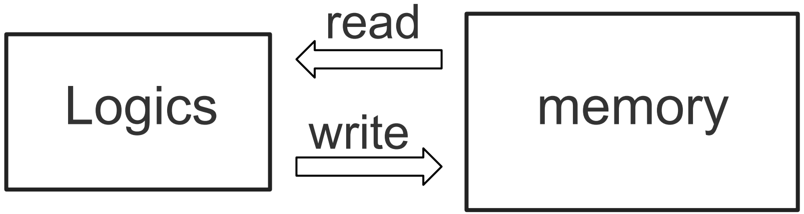
memory

In-Place Matrix Transposition



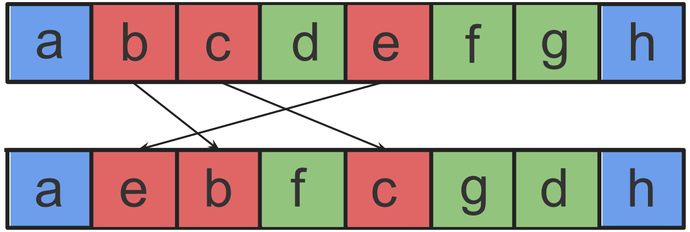
a	b	c	d	e	f	g	h
a	e	b	f	c	g	d	h

(b,c,e) → (e,b,c)
(d,f,g) → (f,g,d)

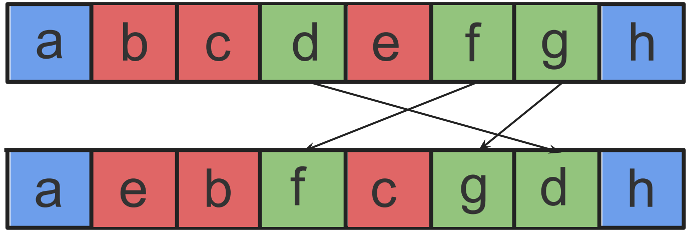


- Need to read multiple entries from the memory, permute them and write them back.
- This operation should be performed efficiently with minimal memory access cost.

In-Place Matrix Transposition



Step 1



Step 2

- The search for optimal swapping patterns that minimize permutations is a well-established problem in mathematics.

Norman Brenner, "Algorithm 467: matrix transposition in place," ACM Transactions on Mathematical Software 16 (11), p. 692-694 (1973). doi:10.1145/355611.362542

Topics

- Matrix Multiplication with Transposition
- Hardware design for Nonlinear Blocks
- System optimization of LLMs
- Popular transformer accelerator design
    - SpAtten
    - EdgeBert
    - Olive

Implementation of Nonlinear Operations: Softmax

- Softmax operations are heavily adopted in the transformer.

$$s_i = \frac{e^{z_i}}{\sum_{j=0}^{j=N-1} e^{z_j}} \quad For\ i = 1, 2, \cdots, N$$

- For positive z with INT representation, we can approximate the values of e^z using the following derivations:

$$e^z = 2^{z\ log_2 e} = 2^{u+v} \qquad log_2 e \approx 1.0111_2$$

$$z\ log_2 e \approx z + (z >> 2) + (z >> 3) + (z >> 4)$$

- To compute 2^{u+v}, we can perform shift and multiplication:

$$e^{z'} = 2^{u+v} \approx 2^u(1 + v/2)$$

u and v are the integer and fractional part of the exponent, v/2 is the mantissa, u is the exponent

NYU SAI LAB

Taylor Approximation

- A Taylor series is a series expansion of a function about a point. A one-dimensional Taylor series is an expansion of a real function f(x) about a point x=a is given by:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \ldots.$$

- For small v, e^v can be approximated as:

$$e^v \approx 1 + \frac{v}{2} \qquad 2^v \approx 1 + \frac{v}{2} \qquad log(1+x) \approx x$$

Xia, Tianhua, and Sai Qian Zhang. "Softmax Acceleration with Adaptive Numeric Format for both Training and Inference." *arXiv preprint arXiv:2311.13290* (2023).

Taylor Approximation

- A Taylor series is a series expansion of a function about a point. A one-dimensional Taylor series is an expansion of a real function f(x) about a point x=a is given by:

$$f(x) = \boxed{f(a) + f'(a)(x-a)} + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \ldots.$$

- For small v, e^v can be approximated as:

$$e^v \approx 1 + \frac{v}{2} \qquad 2^v \approx 1 + \frac{v}{2} \qquad log(1+x) \approx x$$

NYU SAI LAB

Xia, Tianhua, and Sai Qian Zhang. "Softmax Acceleration with Adaptive Numeric Format for both Training and Inference." *arXiv preprint arXiv:2311.13290* (2023).

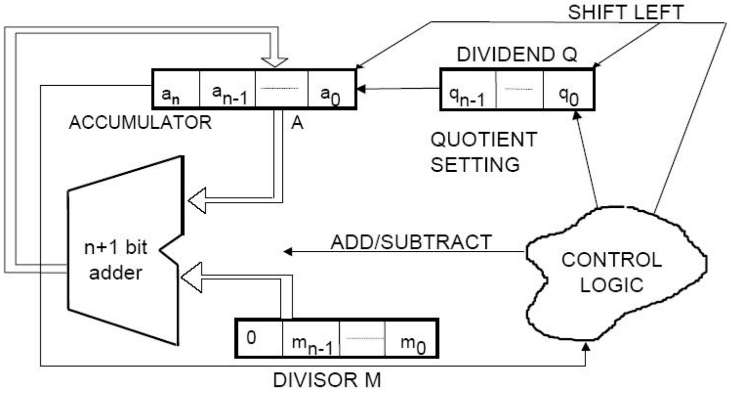
Division

- To implement division operation with FP format, we can always apply the following derivations:

$$\frac{a}{b} = 2^{e_a}(1+m_a)/2^{e_b}(1+m_b) = 2^{e_a-e_b+log_2(1+m_a)-log_2(1+m_b)}$$

$$\approx 2^{e_a-e_b+m_a-m_b} \approx 2^{e_a-e_b}(1+\frac{m_a+m_b}{2})$$

- For INT division, we can also implement the hardware divisor.



NYU SAI LAB

Implementation of Nonlinear Operations: LayerNorm

- For the input vector z, the normalization operation requires to computes its mean and variance, then the intermediate results are scaled with some predefined values.

$$s = \alpha \frac{z - \mu_z}{\sigma_z} + \beta \qquad \mu_z = \frac{\sum_i z_i}{N} \qquad \sigma_z = \sqrt{\frac{\sum_i (z_i - \mu_z)^2}{N}}$$

- Most of the operations are supported, the inverse of square root can be computed as follows:

$$y = \frac{1}{\sqrt{x}} \qquad \log_2(y) = -\frac{1}{2}\log_2(x)$$

Implementation of Nonlinear Operations: LayerNorm

- Most of the operations are supported, the inverse of square root can be computed as follows:

$$y = \frac{1}{\sqrt{x}} \qquad \log_2(y) = -\frac{1}{2}\log_2(x)$$

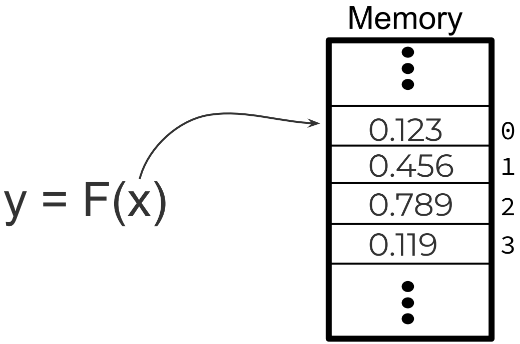
$$x = 2^{E_x - Q}(1 + M_x/2^L) \quad \begin{aligned}\log_2 x &= E_x - Q + \log_2(1 + M_x/2^L) \\ &\approx E_x - Q + M_x/2^L + \sigma_x\end{aligned}$$

- Q is the bias, Ex and Mx are the binary representations of the exponent and mantissa, respectively.

NYU SAI LAB

Table Lookup

- For other complicated nonlinear functions, we can always precompute the results and store them in the buffer.



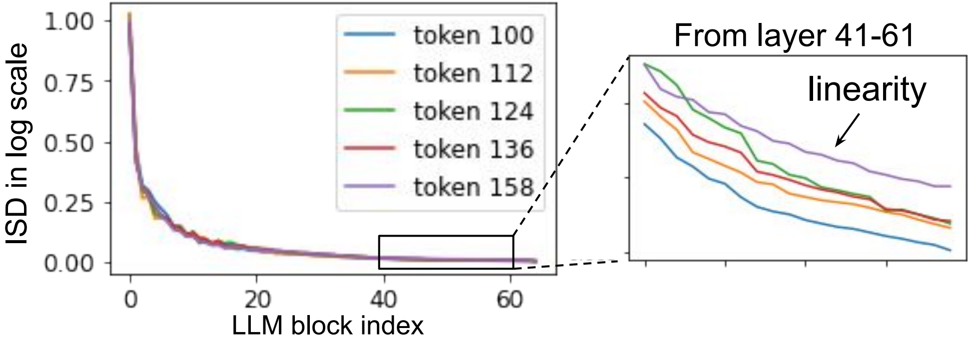
- However, this will inevitably lead to additional memory access cost and footprint.

HAAN: LayerNorm Accelerator

Layer Normalization:

$$s = \alpha \frac{z - \mu_z}{\boxed{\sigma_z}} + \beta$$

Computing the inverse of standard deviation of costly



- Exploit correlation in input statistics across layers.
- Skip redundant computations and estimate normalization statistics.

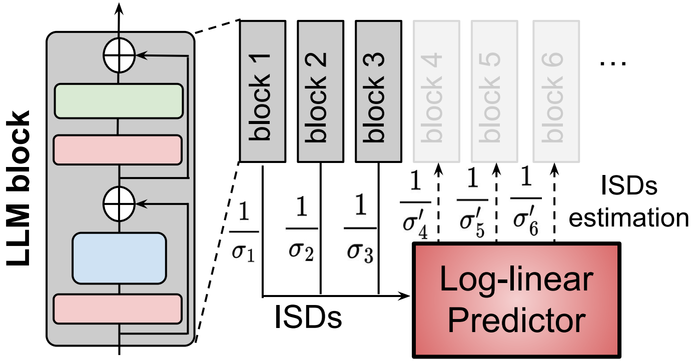
Peng, Tianfan, et al. "HAAN: A Holistic Approach for Accelerating Normalization Operations in Large Language Models." *arXiv preprint arXiv:2502.11832* (2025).

HAAN: LayerNorm Accelerator

Layer Normalization:

$$\boldsymbol{s} = \alpha \frac{\boldsymbol{z} - \mu_{\boldsymbol{z}}}{\boxed{\sigma_{\boldsymbol{z}}}} + \beta$$

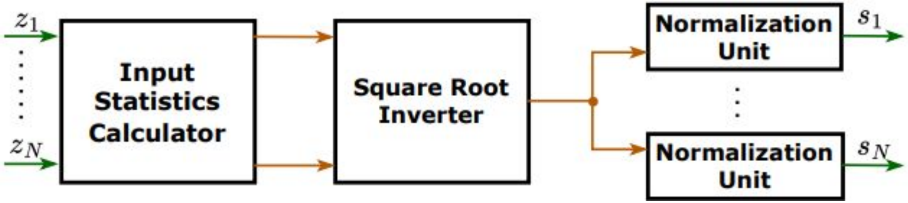
Computing the inverse of standard deviation of costly

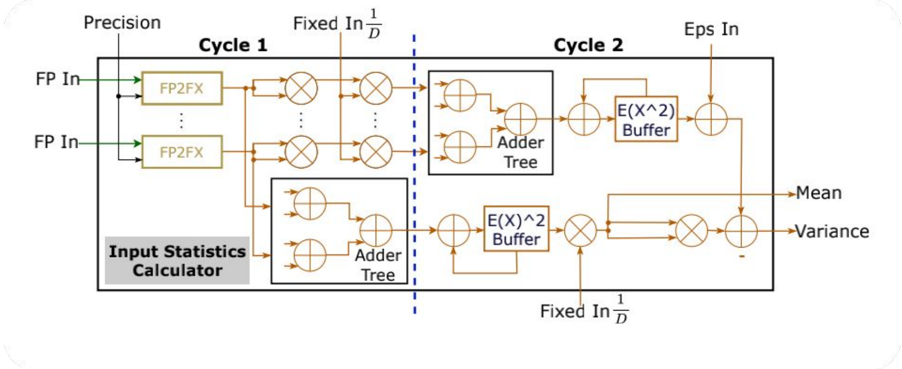


- Exploit correlation in input statistics across layers.
- Skip redundant computations and estimate normalization statistics.

NYU SAI LAB

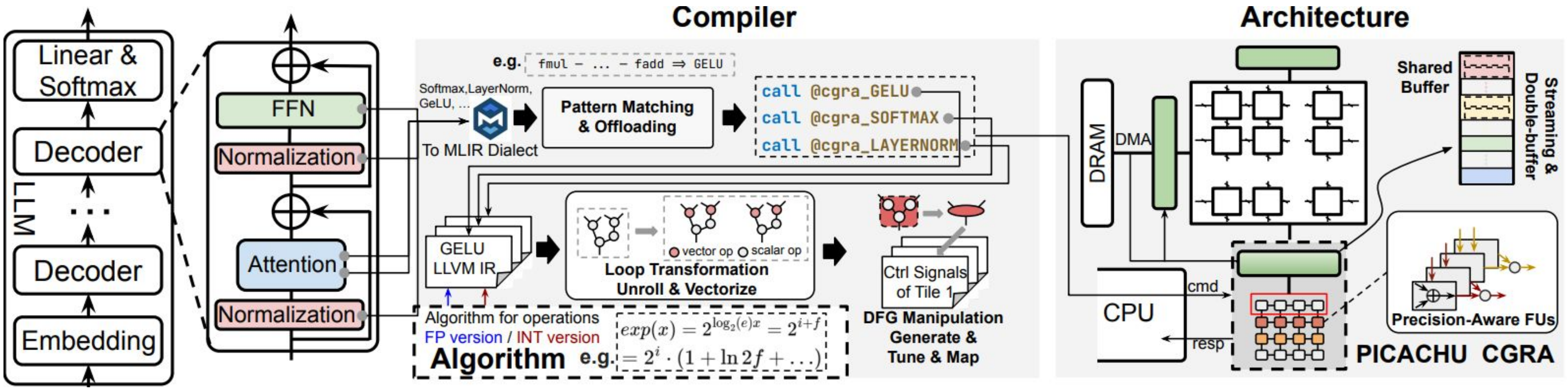
HAAN: LayerNorm Accelerator





- **Overall Architecture**
  - Input Statistics Calculator.
  - Square Root Inverter.
  - Normalization Unit.
- **Input Statistics Calculator**
  - Compute mean and variance.
  - Parallel processing to reduce latency.
- **Square Root Inverter**
  - Approximate inverse square root using Newton's method.
  - Support for layer skipping.

PICACHU

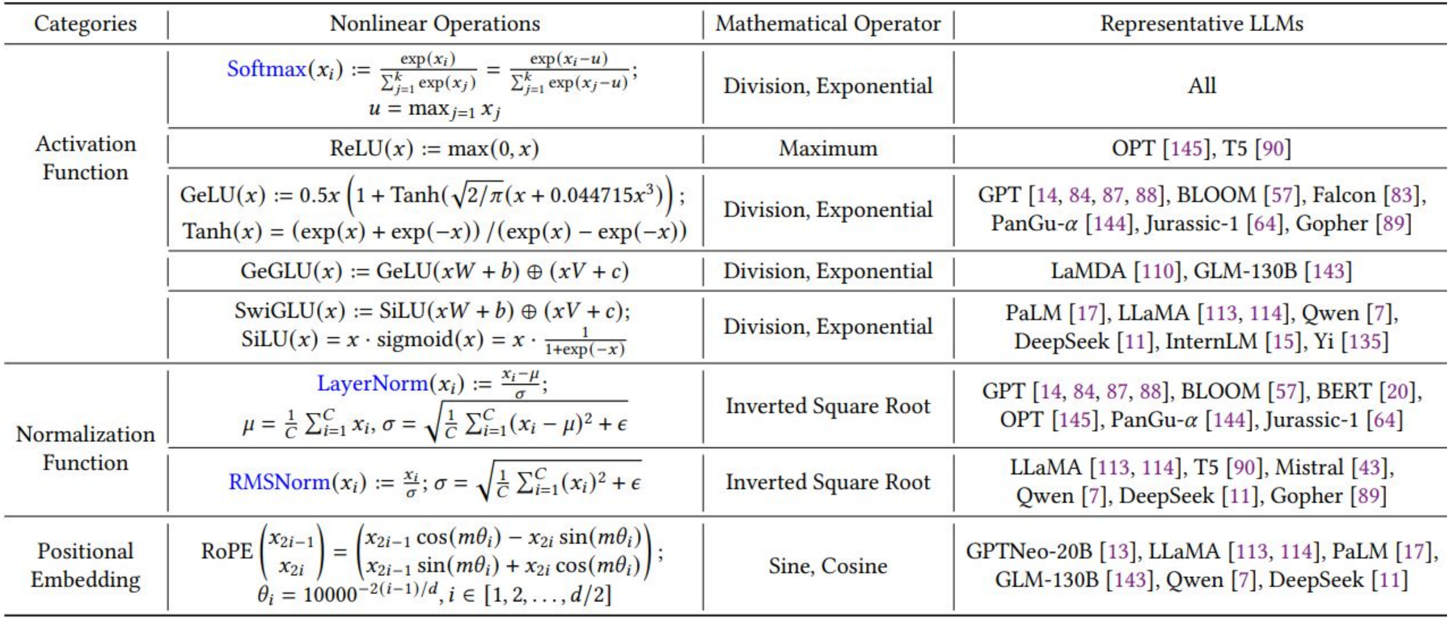


- PICACHU is a plug-in coarse-grained reconfigurable accelerator tailored to efficiently handle nonlinear operations by using custom algorithms and a dedicated compiler toolchain.

NYU SAI LAB

Qin, Jiajun, et al. "PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs." Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 2025.

PICACHU



Categories	Nonlinear Operations	Mathematical Operator	Representative LLMs
Activation Function	Softmax$(x_i) := \frac{\exp(x_i)}{\sum_{j=1}^{k}\exp(x_j)} = \frac{\exp(x_i-u)}{\sum_{j=1}^{k}\exp(x_j-u)}$; $u = \max_{j=1} x_j$	Division, Exponential	All
	$\mathrm{ReLU}(x) := \max(0, x)$	Maximum	OPT [145], T5 [90]
	$\mathrm{GeLU}(x) := 0.5x\left(1+\mathrm{Tanh}(\sqrt{2/\pi}(x+0.044715x^3))\right)$; $\mathrm{Tanh}(x) = (\exp(x)+\exp(-x))/(\exp(x)-\exp(-x))$	Division, Exponential	GPT [14, 84, 87, 88], BLOOM [57], Falcon [83], PanGu-α [144], Jurassic-1 [64], Gopher [89]
	$\mathrm{GeGLU}(x) := \mathrm{GeLU}(xW+b) \oplus (xV+c)$	Division, Exponential	LaMDA [110], GLM-130B [143]
	$\mathrm{SwiGLU}(x) := \mathrm{SiLU}(xW+b) \oplus (xV+c)$; $\mathrm{SiLU}(x) = x \cdot \mathrm{sigmoid}(x) = x \cdot \frac{1}{1+\exp(-x)}$	Division, Exponential	PaLM [17], LLaMA [113, 114], Qwen [7], DeepSeek [11], InternLM [15], Yi [135]
Normalization Function	LayerNorm$(x_i) := \frac{x_i-\mu}{\sigma}$; $\mu = \frac{1}{C}\sum_{i=1}^{C} x_i, \sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i-\mu)^2+\epsilon}$	Inverted Square Root	GPT [14, 84, 87, 88], BLOOM [57], BERT [20], OPT [145], PanGu-α [144], Jurassic-1 [64]
	RMSNorm$(x_i) := \frac{x_i}{\sigma}; \sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i)^2+\epsilon}$	Inverted Square Root	LLaMA [113, 114], T5 [90], Mistral [43], Qwen [7], DeepSeek [11], Gopher [89]
Positional Embedding	$\mathrm{RoPE}\begin{pmatrix}x_{2i-1}\\ x_{2i}\end{pmatrix} = \begin{pmatrix}x_{2i-1}\cos(m\theta_i) - x_{2i}\sin(m\theta_i)\\ x_{2i-1}\sin(m\theta_i) + x_{2i}\cos(m\theta_i)\end{pmatrix}$; $\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \ldots, d/2]$	Sine, Cosine	GPTNeo-20B [13], LLaMA [113, 114], PaLM [17], GLM-130B [143], Qwen [7], DeepSeek [11]

- All nonlinear operations within LLM can be broken down into various mathematical operators.

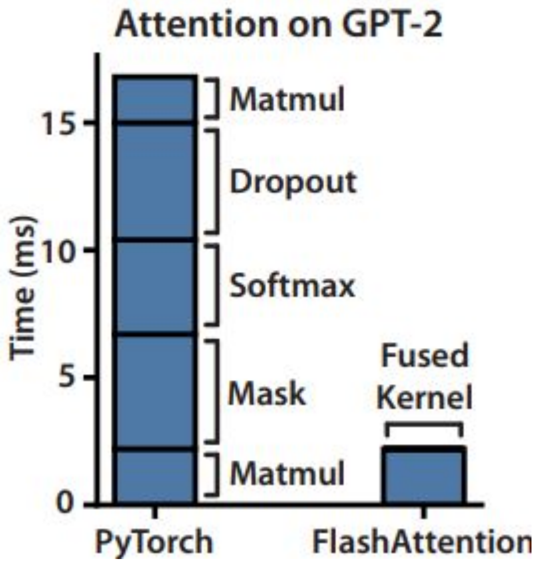
Qin, Jiajun, et al. "PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs." Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 2025.

Topics

- Matrix Multiplication
- Hardware design for Nonlinear Blocks
- System optimization of LLMs
- Popular transformer accelerator design
  - SpAtten
  - EdgeBert
  - Olive

Flashattention

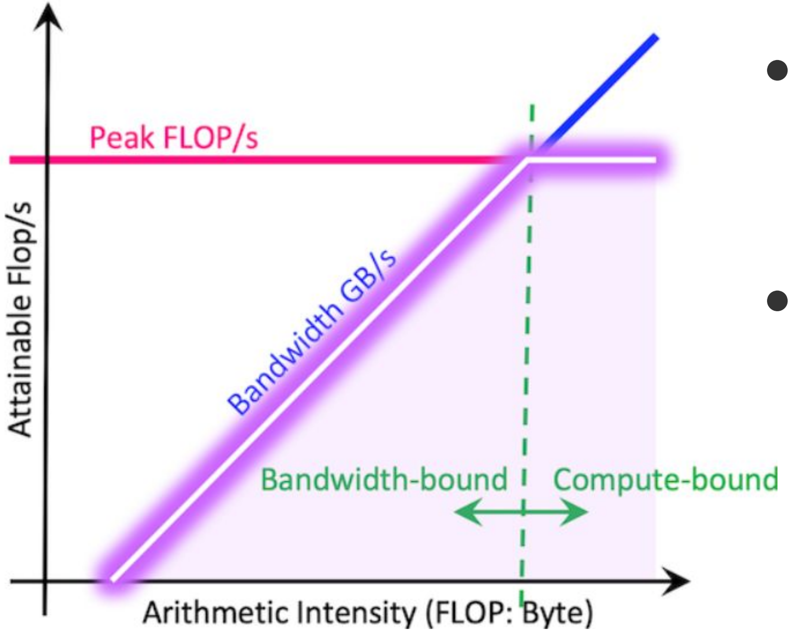
- Most of the operations are bottlenecked by memory speed.
- A new attention algorithm that computes exact attention with far fewer memory accesses.
- The main goal is to avoid reading and writing the attention matrix to and from memory.
- Flashattention enables to compute the softmax reduction without access to the whole input.



7.6x reduction on GPU latency

Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." *Advances in Neural Information Processing Systems* 35 (2022): 16344-16359.

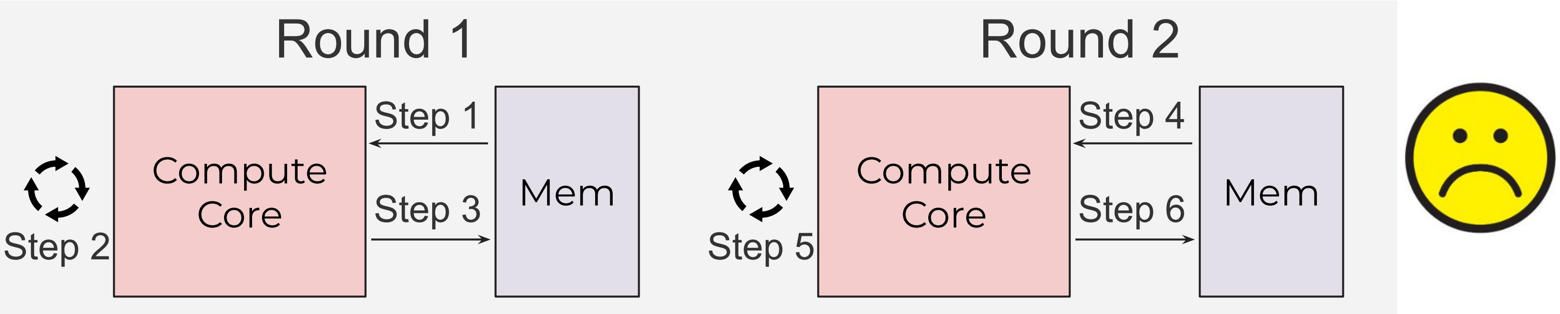
Flashattention

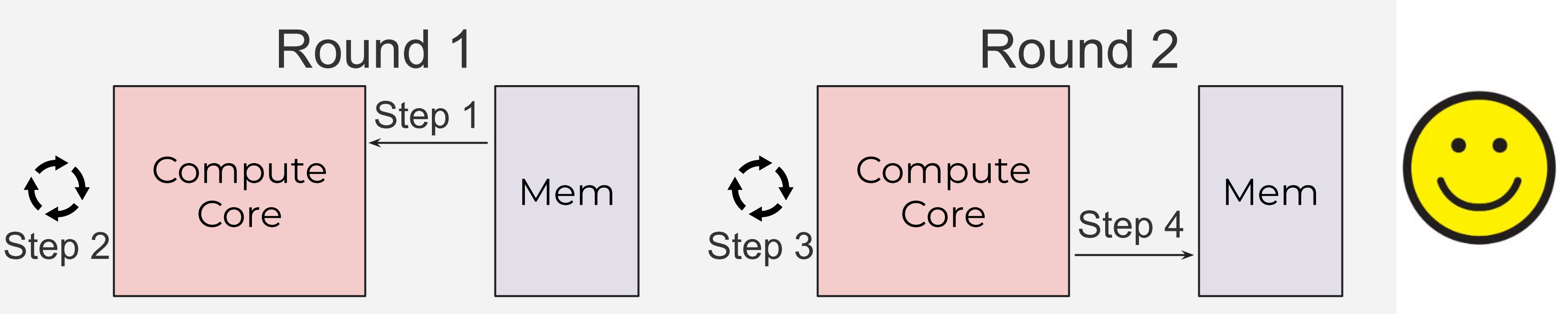


- Due to the heavy involvement of attention mechanism, transformers are memory-bound rather than compute bound.
- Kernel fusion: if there are multiple operations applied to the same input, the input can be loaded once from HBM, instead of multiple times for each operation.

Flashattention

$Y = S(QK^T) \times V$



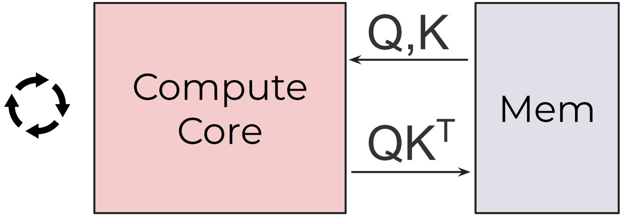


NYU SAI LAB

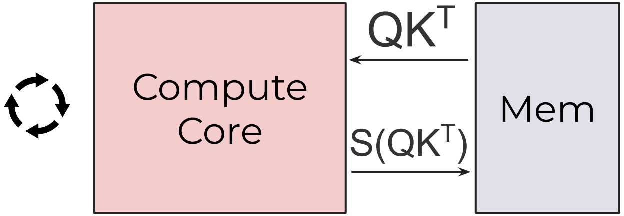
Flashattention

$Y = S(QK^T) \times V$

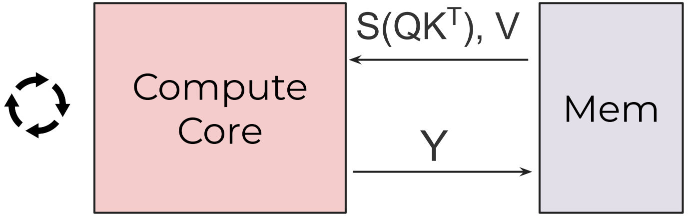
Round 1



Round 2



Round 3



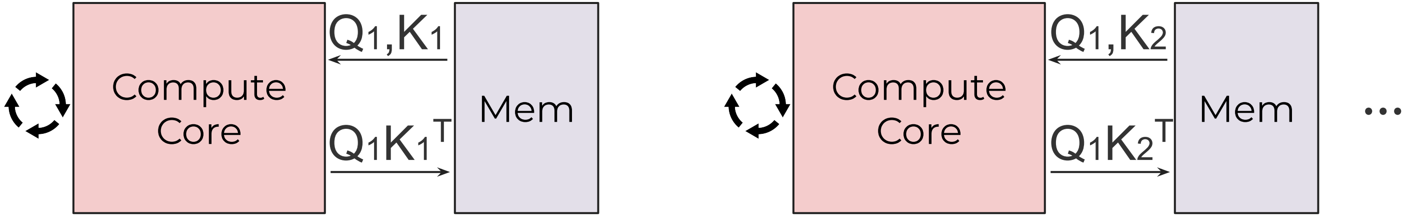
- The computation of QK^T must be all finished before computing softmax.
- This will lead to multiple rounds of memory access.

$$s_i = \frac{e^{z_i}}{\sum_{j=0}^{j=N-1} e^{z_j}} \quad For\ i = 1, 2, \cdots, N$$

Flashattention

$Y = S(QK^T) \times V$

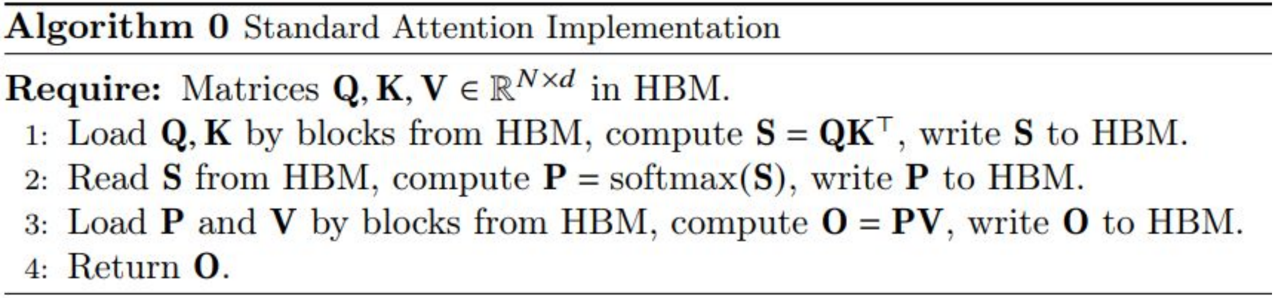
Round 1



- In practice, this operation will be executed in tiles.

NYU SAI LAB

Flashattention



Algorithm 0 Standard Attention Implementation

```
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.
Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
Return $\mathbf{O}$.
```

- Softmax and linear layers are computed separately.
- Flashattention splits the inputs Q, K, V into tiles, then compute the attention output with respect to those blocks.

Flashattention

- Softmax operation can be performed as:

$$m(x) := \max_i \; x_i, \quad f(x) := \begin{bmatrix} e^{x_1 - m(x)} & \ldots & e^{x_B - m(x)} \end{bmatrix}$$

$$\ell(x) := \sum_i f(x)_i, \quad \mathrm{softmax}(x) := \frac{f(x)}{\ell(x)}$$

- We can fuse the softmax with the linear layer by doing follows:

$$m(x) = m(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = \max(m(x^{(1)}), m(x^{(2)}))$$

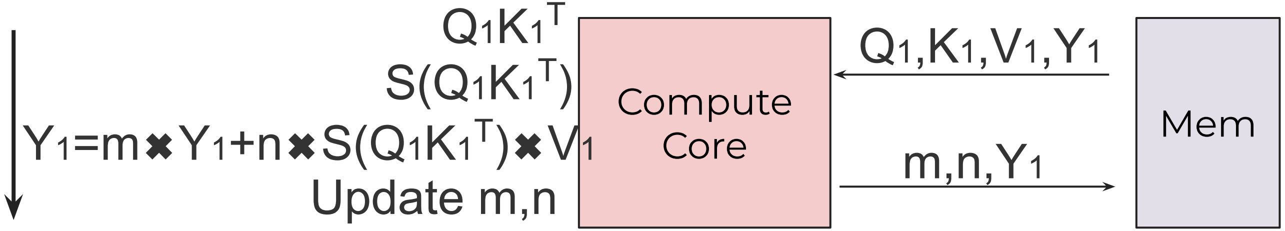
$$f(x) = \begin{bmatrix} e^{m(x^{(1)}) - m(x)} f(x^{(1)}) & e^{m(x^{(2)}) - m(x)} f(x^{(2)}) \end{bmatrix}$$

$$\ell(x) = \ell(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = e^{m(x^{(1)}) - m(x)} \ell(x^{(1)}) + e^{m(x^{(2)}) - m(x)} \ell(x^{(2)}) \quad \mathrm{softmax}(x) = \frac{f(x)}{\ell(x)}$$

Flashattention

$S(QK^T) \times V$

Round 1



$Q_1K_1^T$

$S(Q_1K_1^T)$

$Y_1 = m \times Y_1 + n \times S(Q_1K_1^T) \times V_1$

Update m,n

Compute Core

Q_1, K_1, V_1, Y_1

m, n, Y_1

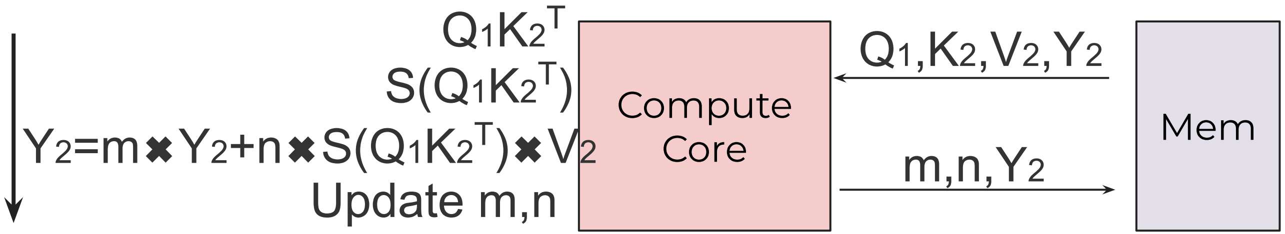
Mem

$$s_i = \frac{e^{z_i}}{\sum_{j=0}^{j=N-1} e^{z_j}} \quad For\ i = 1, 2, \cdots, N$$

Flashattention

$S(QK^T) \times V$

Round 2



$Q_1K_2^T$

$S(Q_1K_2^T)$

$Y_2=m \times Y_2+n \times S(Q_1K_2^T) \times V_2$

Update m,n

Compute Core

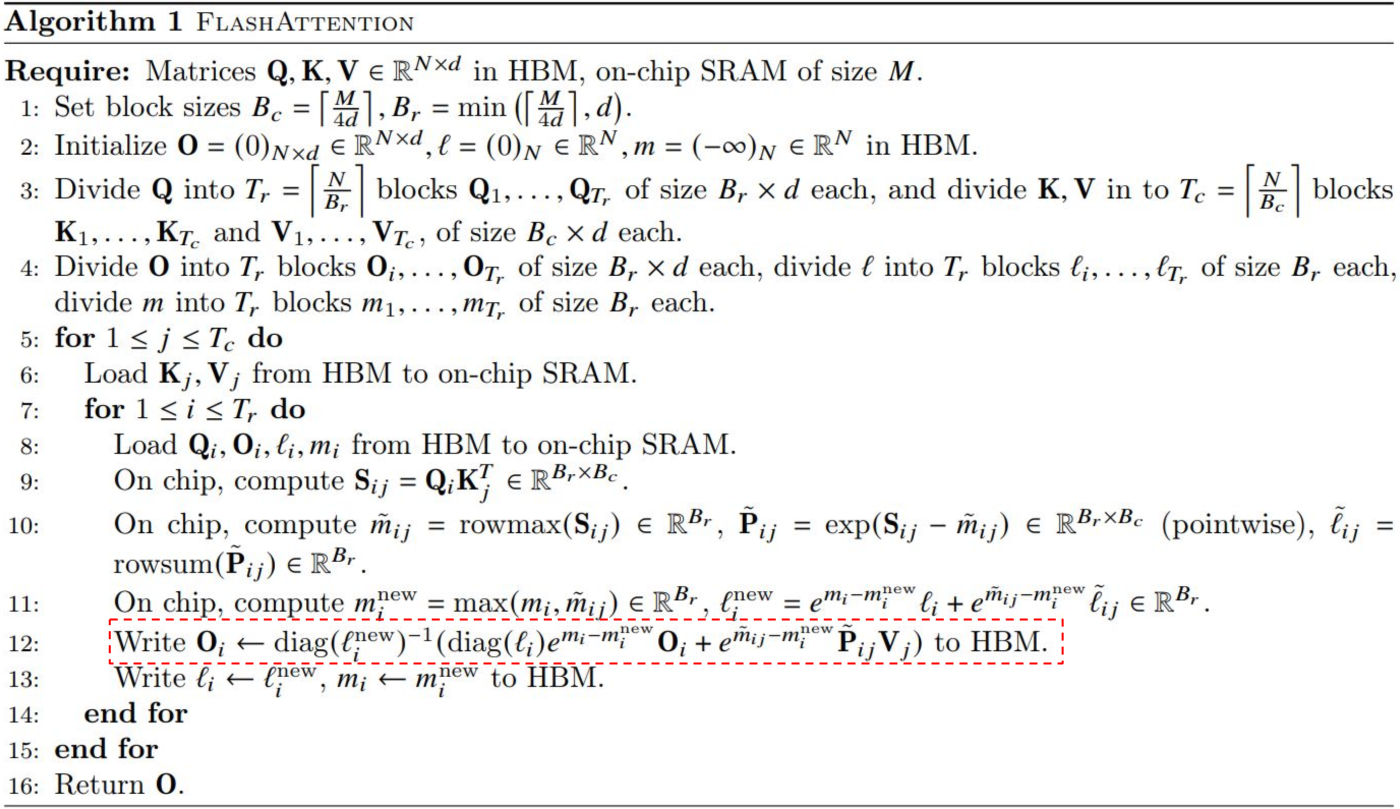
Q_1,K_2,V_2,Y_2

m,n,Y_2

Mem

$$s_i = \frac{e^{z_i}}{\sum_{j=0}^{j=N-1} e^{z_j}} \quad For\ i = 1, 2, \cdots, N$$

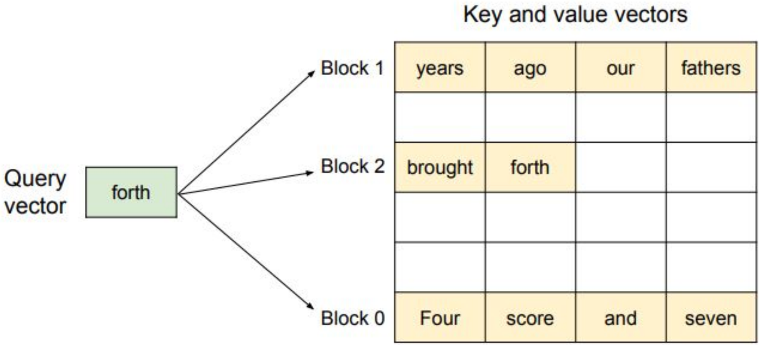
Flashattention

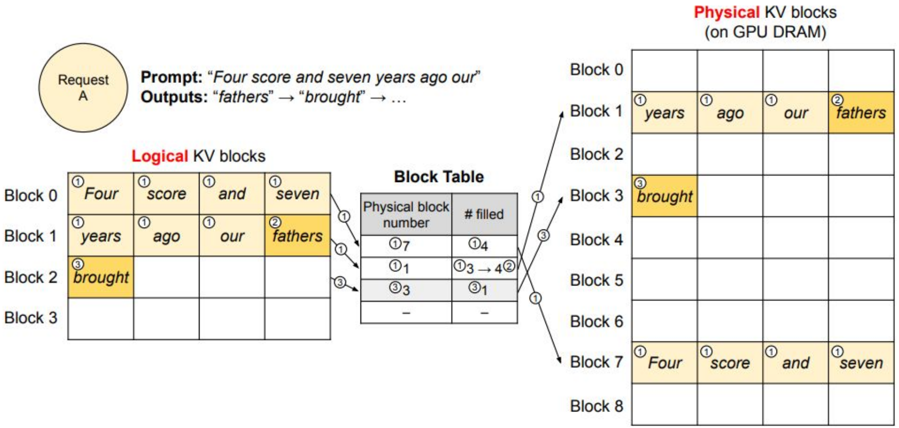


Algorithm 1 FlashAttention

```
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size $M$.
Set block sizes $B_c = \left\lceil \frac{M}{4d} \right\rceil, B_r = \min\left(\left\lceil \frac{M}{4d} \right\rceil, d\right)$.
Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
Divide $\mathbf{O}$ into $T_r$ blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide $\ell$ into $T_r$ blocks $\ell_i, \dots, \ell_{T_r}$ of size $B_r$ each, divide $m$ into $T_r$ blocks $m_1, \dots, m_{T_r}$ of size $B_r$ each.
for $1 \le j \le T_c$ do
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    for $1 \le i \le T_r$ do
        Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
        On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
        On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
        On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
        Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\text{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
        Write $\ell_i \leftarrow \ell_i^{\text{new}}$, $m_i \leftarrow m_i^{\text{new}}$ to HBM.
    end for
end for
Return $\mathbf{O}$.
```

Paged Attention





Prompt: "*Four score and seven years ago our*"
Outputs: "*fathers*" → "*brought*" → …

- An LLM serving system that achieves (1) near-zero waste in KV cache memory and (2) flexible sharing of KV cache within and across requests to further reduce memory usage.

NYU SAI LAB

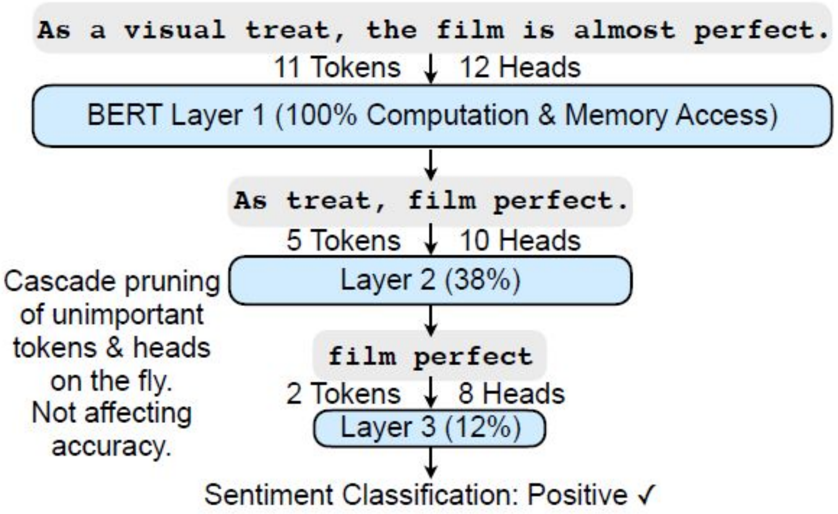
Kwon, Woosuk, et al. "Efficient memory management for large language model serving with paged attention." *Proceedings of the 29th Symposium on Operating Systems Principles*. 2023.

Topics

- Matrix Multiplication
- Hardware design for Nonlinear Blocks
- System optimization of LLMs
- Popular transformer accelerator design
  - SpAtten
  - EdgeBert
  - Olive

NYU SAI LAB

SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning



As a visual treat, the film is almost perfect.

11 Tokens ↓ 12 Heads

BERT Layer 1 (100% Computation & Memory Access)

As treat, film perfect.

5 Tokens ↓ 10 Heads

Layer 2 (38%)

film perfect

2 Tokens ↓ 8 Heads

Layer 3 (12%)

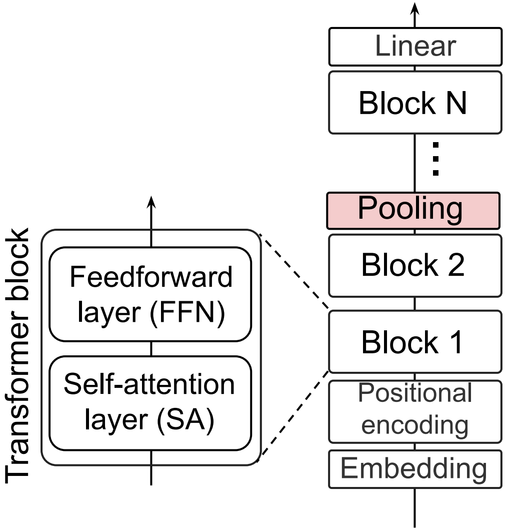
Sentiment Classification: Positive ✓

Cascade pruning of unimportant tokens & heads on the fly. Not affecting accuracy.

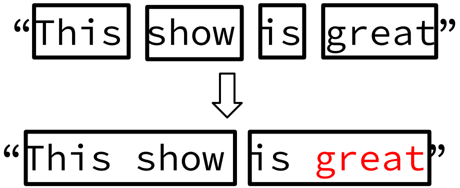
- Not all the tokens nor heads are necessary to produce the final results.
- SpAtten is an efficient algorithm-architecture co-design that leverages token sparsity, head sparsity, and quantization opportunities to reduce the attention computation and memory access.

Wang, Hanrui, Zhekai Zhang, and Song Han. "Spatten: Efficient sparse attention architecture with cascade token and head pruning." *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021.

Token Merging



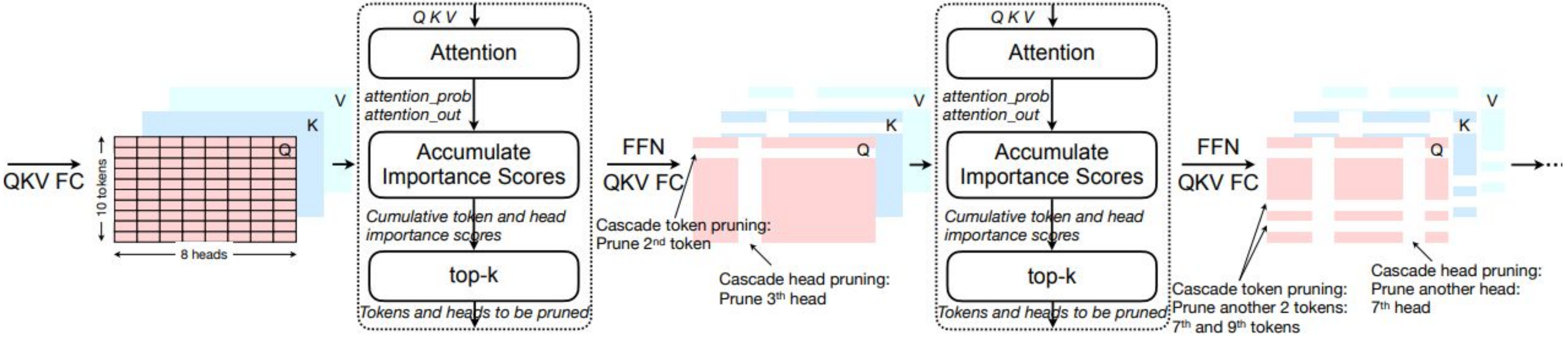
Pooling layer will reduce the number of tokens.



- We can reduce the number of tokens by merging them together.

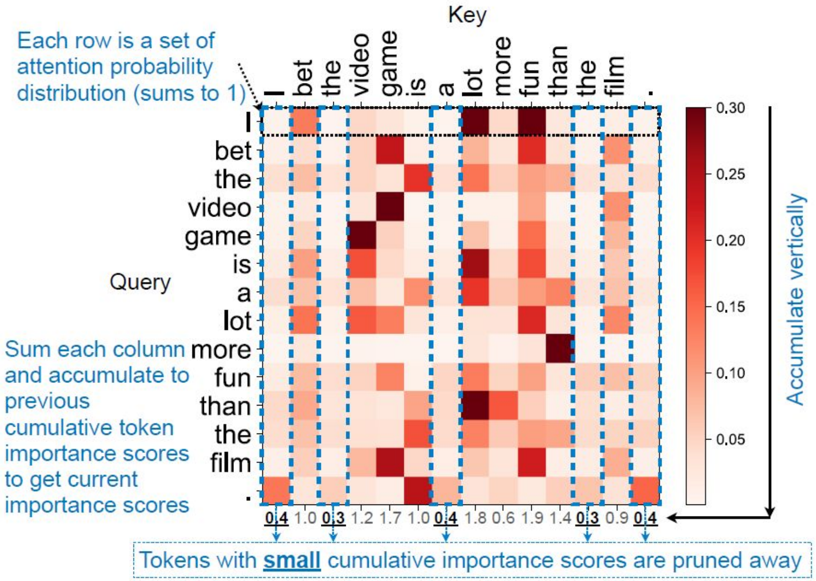
Bolya, Daniel, et al. "Token merging: Your vit but faster." *arXiv preprint arXiv:2210.09461* (2022).

SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning



- Cascade token pruning removes redundant tokens and corresponding entire Q K V vectors according to the cumulative token importance scores computed from attention prob.
- Cascade head pruning removes unimportant heads and corresponding chunks in all Q K V vectors according to the cumulative head important scores computed from attention out.

SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning



Each row is a set of attention probability distribution (sums to 1)

Sum each column and accumulate to previous cumulative token importance scores to get current importance scores

Tokens with **small** cumulative importance scores are pruned away

- For each input, the summation of each columns indicates the importance of this token, we can remove the unimportant tokens accordingly.
- Similarly, we can compute the importance of each heads.

NYU SAI LAB

SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning

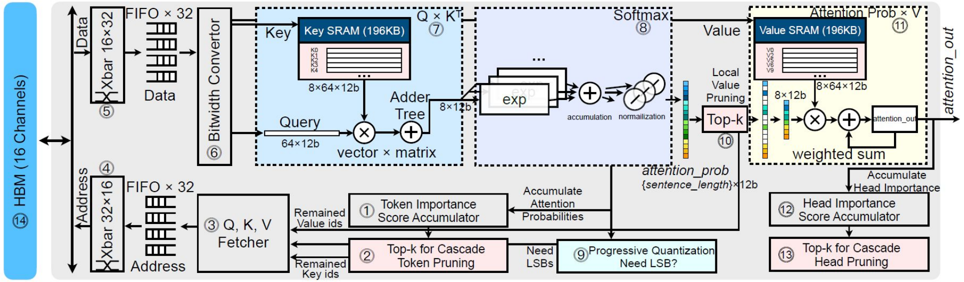
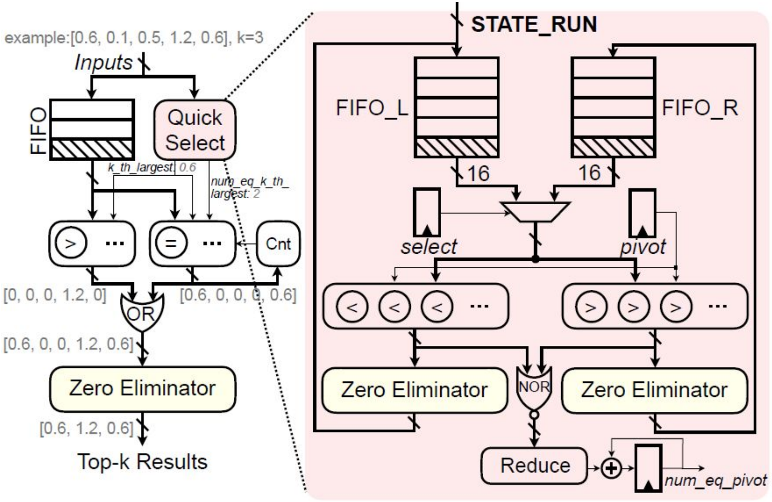


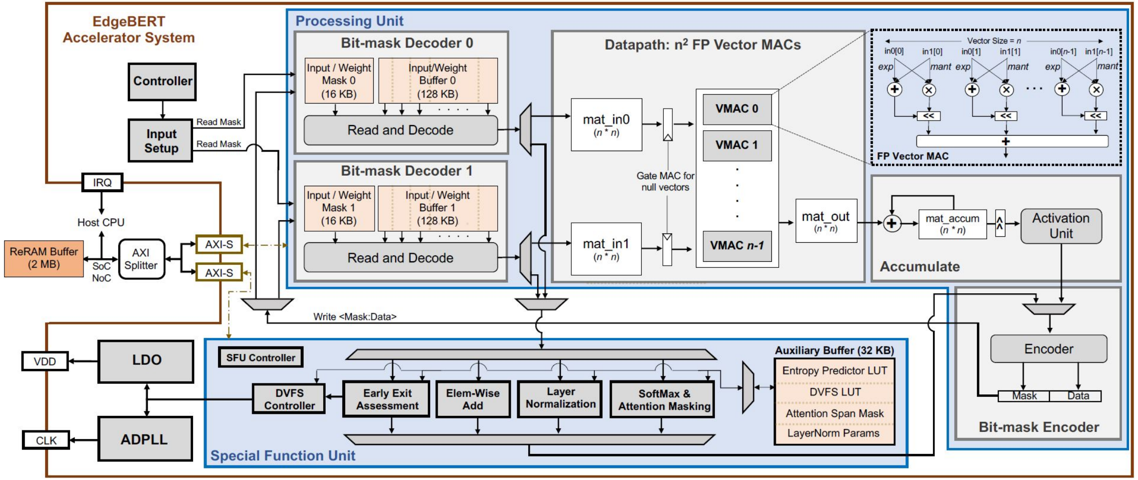
Fig. 8. SpAtten Architecture Overview. Modules on the critical path (6,7,8,10,11) are fully pipelined to maximize the throughput.

- This paper also proposed novel architecture to perform top-k extraction with high parallelism.



NYU SAI LAB

EdgeBert



$$Energy \propto \alpha C V_{DD}^2 N_{cycles}$$

- EdgeBERT is a novel algorithm-hardware codesign approach to enable latency-bound NLP workloads on resource-constrained embedded devices.
- EdgeBERT dynamically tunes frequency and voltage settings to optimize the trade-off between accuracy, latency, and energy consumption.

NYU SAI LAB

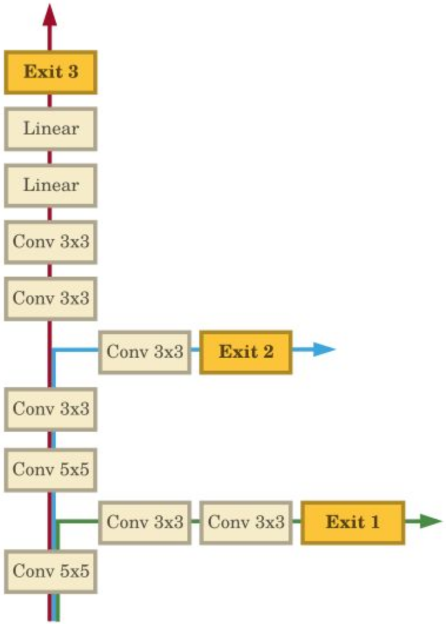
Tambe, Thierry, et al. "Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference." *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 2021.

Early Exit Mechanism

- EdgeBERT employs entropy-based early exit predication in order to perform dynamic voltage-frequency scaling (DVFS), at a sentence granularity, for minimal energy consumption while adhering to a prescribed target latency.

- During Inference, a confidence score is computed at each exit point, if greater than a predefined threshold, then the output is computed locally, leading to a faster inference.

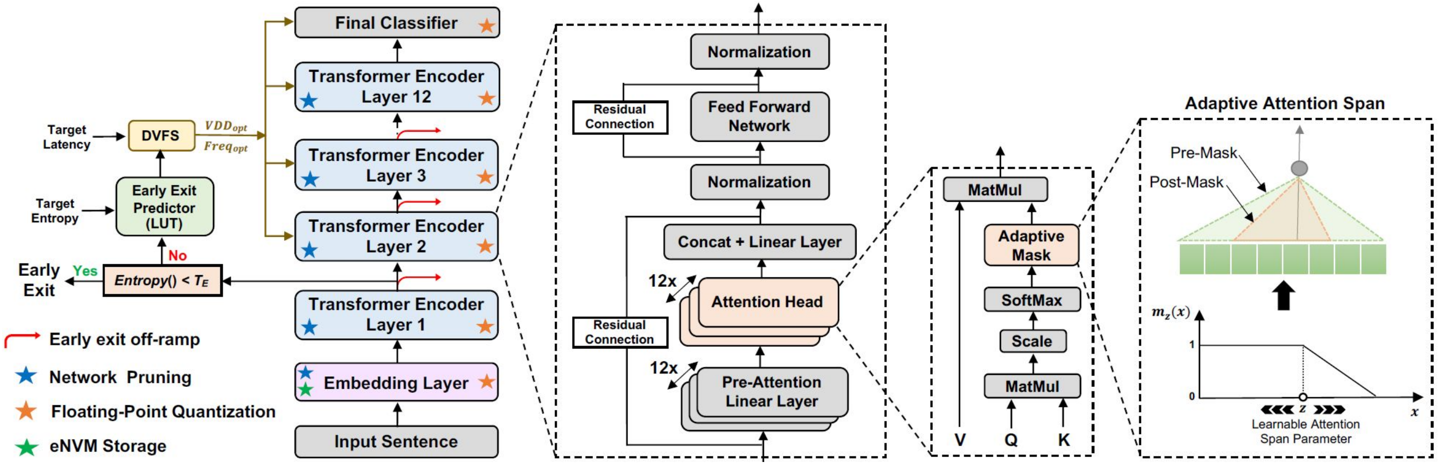
- The confidence score is defined as: $\text{entropy}(\boldsymbol{y}) = \sum_{c \in \mathcal{C}} y_c \log y_c,$



NYU SAI LAB

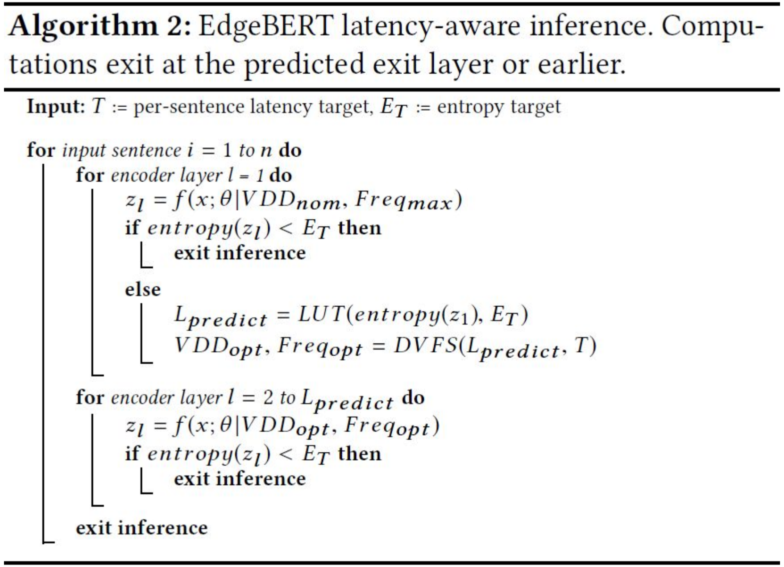
Teerapittayanon, Surat, Bradley McDanel, and Hsiang-Tsung Kung. "Branchynet: Fast inference via early exiting from deep neural networks." *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016.

Other Tricks for Efficiency



- Pruning and Quantization techniques are also adopted.

Latency Aware Inference



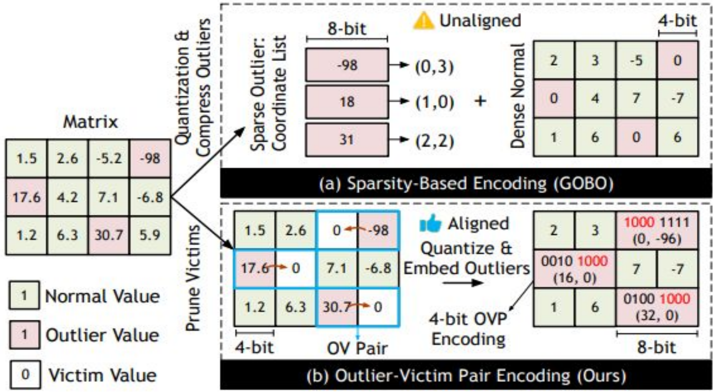
Algorithm 2: EdgeBERT latency-aware inference. Computations exit at the predicted exit layer or earlier.

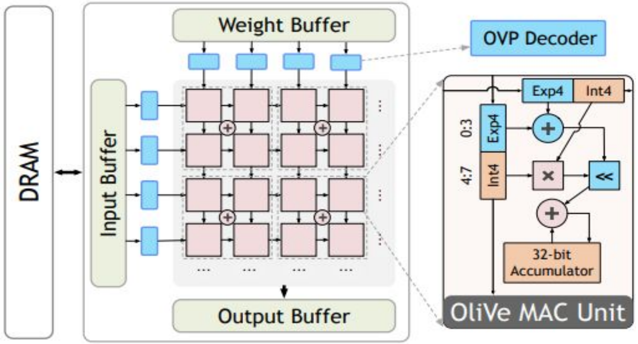
```
Input: $T$ := per-sentence latency target, $E_T$ := entropy target

for input sentence $i = 1$ to $n$ do
    for encoder layer $l = 1$ do
        $z_l = f(x; \theta | VDD_{nom}, Freq_{max})$
        if $entropy(z_l) < E_T$ then
            exit inference
        else
            $L_{predict} = LUT(entropy(z_1), E_T)$
            $VDD_{opt}, Freq_{opt} = DVFS(L_{predict}, T)$
    for encoder layer $l = 2$ to $L_{predict}$ do
        $z_l = f(x; \theta | VDD_{opt}, Freq_{opt})$
        if $entropy(z_l) < E_T$ then
            exit inference
    exit inference
```

- The entropy result of the first layer is sent to a trained classifier (LUT-based) to predict which following encoder layer should early exit.
- The voltage and frequency is scaled down to proper energy-optimal setting for the rest of encoder layers while meeting the latency target for each sentence.
- This scheme produces a quadratic reduction in the accelerator power consumption.
- To realize fast per-sentence DVFS, the on-chip DVFS system is developed and integrated within EdgeBERT

OliVe: Accelerating LLMs via Hardware-friendly Outlier-Victim Pair Quantization



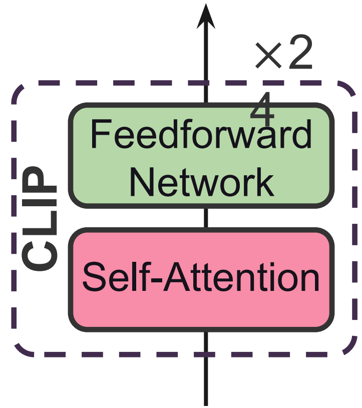


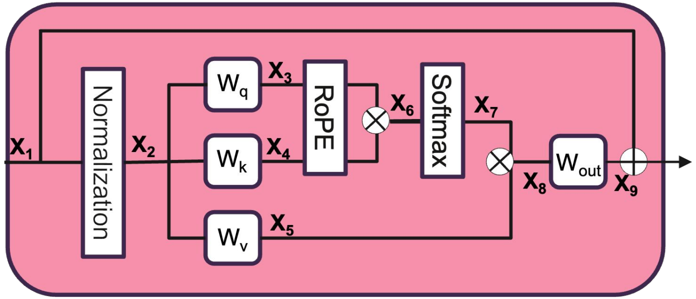
- Recent studies show when the model size exceeds a threshold (e.g., 6 billion), the model performance is vulnerable to only a tiny fraction (< 0.1%) of outliers, whose values are much more significant than normal values.
- Olive adopts a hybrid quantization scheme and handles outlier values locally using a separate quantization scheme.

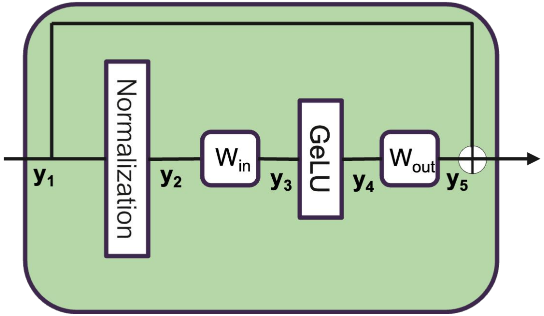
NYU SAI LAB

Guo, Cong, et al. "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization." *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 2023.

Outlier within LLMs



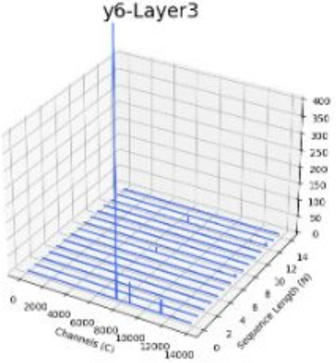


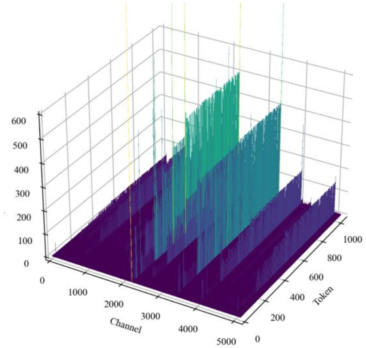


Radford, Alec, et al. "Learning transferable visual models from natural language supervision." *International conference on machine learning*. PMLR, 2021.

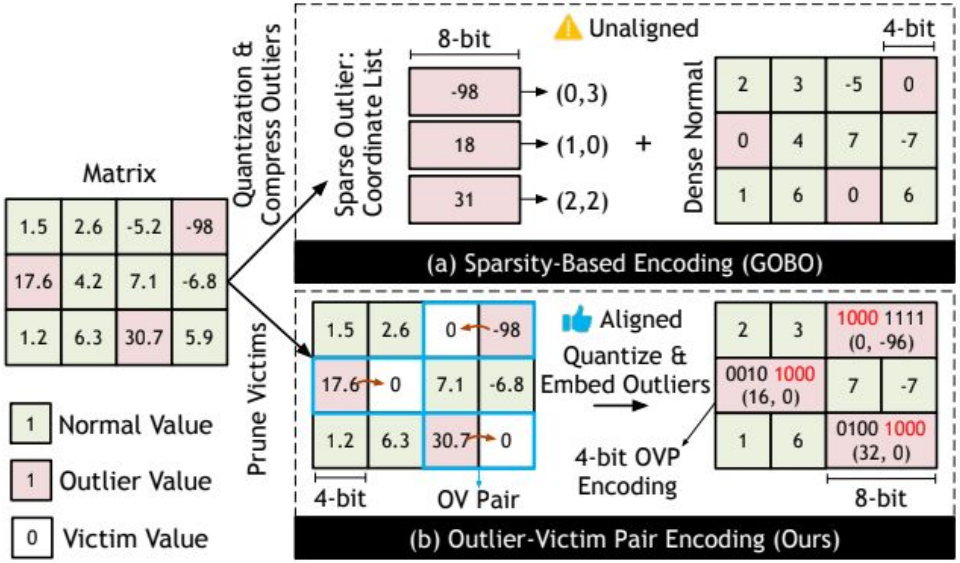
Types of Outlier

- Massive Activation:
  - For an activation matrix A, an massive activation is an element Aij within it that satisfies:
  - Aij > η✕mean(|A|)
  - Aij > γ
  - η=300, γ=50
- Channelwise Outlier:
  - mean(Ai) > η✕std(A) +mean(|A|)
  - std(Ai) < β
  - η=3, β=0.6





OliVe: Accelerating LLMs via Hardware-friendly Outlier-Victim Pair Quantization



- In this work, we aim to design an architecture to handle outliers in a localized way with high hardware efficiency. Post quantization training is adopted.
- To better quantize the outlier, one number near the outlier are sacrificed and set to 0, then two set of bits are used to encode the outlier.

Presentations

- Efficient Memory Management for Large Language Model Serving with PagedAttention (Xiwen Min & Ziyun Cheng)

- Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing (Jishnu Warrier & Ishaan Shivhare)